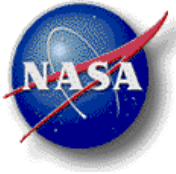


## NCCS Brown Bag Series

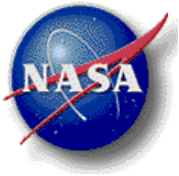


---

# TotalView on Discover: Part 1

## Parallel Debugging

Chongxun (Doris) Pan  
doris.pan@nasa.gov  
May 8, 2012



# Agenda



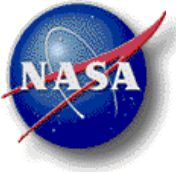
- **Overview**
- Basic Navigation and Control
- Demo 1: Basic Navigation and Control
  - ❖ Using an OpenMP Space weather application
- Debugging MPI Applications
- Demo 2: MPI Debugging:
  - ❖ Using GEOS5-AGCM



## TotalView 8.9.2-2 currently on Discover



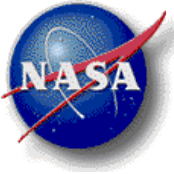
- An interactive tool that lets you debug serial, multi-threaded and multi-processor programs with support for Fortran and C/C++
- The base TotalView debugger (TVD) solution now includes Replay Engine and CUDA debugging support for no additional fee
- Major features:
  - ❖ Parallel debugging: MPI, Pthreads, OpenMP, CUDA
  - ❖ Reverse debugging with **ReplayEngine** 2.1.0-2
  - ❖ Integrated Memory debugging with **MemoryScape** 3.2.2-2
  - ❖ Batch debugging with TVScript and the CLI



# What is ReplayEngine ?



- Reverse Debugging
  - ❖ Capture and deterministically replay execution
  - ❖ Eliminate restart cycle and hard-to-reproduce bugs
  - ❖ Step back and forward by function, line, or instruction
- Major features
  - ❖ No recompilation
  - ❖ Supports both MPI and OpenMP
  - ❖ Setenv TVD\_REPLAY\_TMPDIR to control the directory of saving the history information



# What is MemoryScape?

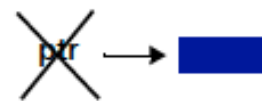


- Runtime Memory Analysis to detect memory bugs
  - ❖ A memory bug is a mistake in heap memory usage
    - ❖ Leaking: failure to free memory
    - ❖ Dangling references: failure to clear pointers
    - ❖ Memory corruption
  - ❖ Use for validation as part of a quality software development process

- Major features

- ❖ No recompilation
- ❖ Supports both MPI and OpenMP

ptr →  *normal allocation*



*leaked memory*



*dangling pointer*



## More Info

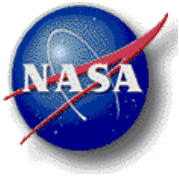


- Details on how to configure TV for your runs and use TV debugging serial, OpenMP, and MPI/OpenMP jobs can also be found in the NCCS Primer:

<http://www.nccs.nasa.gov/primer/computing.html#totalview>

- Short TotalView video tutorials on interesting topics can be found at:

<http://www.roguewave.com/products/totalview/resources/videos.aspx>

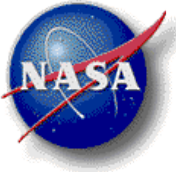


# Agenda



- Overview
- **Basic Navigation and Control**
- Demo 1: Basic Navigation and Control
  - ❖ Using an OpenMP Space weather application
- Debugging MPI Applications
- Demo 2: MPI Debugging:
  - ❖ Using GEOS5-AGCM





## Starting TotalView



- `-g` has to be used for debugging. Using `-g` automatically adds `-O0`
- Load module: `module load tool/tview-8.9.2-2`
- `setenv TVDSVRLAUNCHCMD ssh`
- Launch Totalview GUI via Command line

❖ Normal:

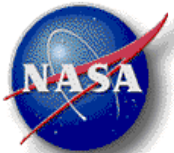
`totalview executable_name [-a executable_args]`

❖ Attach to a running program

`totalview executable_name -pid PID# [-a executable_args]`

❖ Attach to a core file

`totalview executable_name corefile_name [-a executable_args]`

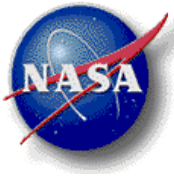


# Starting TotalView



```
discover25:~/cpan2>xsub -I -V -l select=1:ncpus=12,walltime=30:00 -W group_list=k3001
Establishing X forwarding and submitting batch job...
qsub: waiting for job 1790833.borgpbs1 to start
qsub: job 1790833.borgpbs1 ready

borgc002:~/cpan2>module purge
borgc002:~/cpan2>module load comp/intel-12.1.0.233 tool/tview-8.9.2.2
borgc002:~/cpan2>more $PBS_NODEFILE
borgc002
borgc002:~/cpan2>cd /discover/nobackup/cpan2/tmp/K_src/
borgc002:~/K_src>setenv OMP_NUM_THREADS 4
borgc002:~/K_src>which totalview
/usr/local/toolworks/totalview.8.9.2-2/bin/totalview
borgc002:~/K_src>totalview ./my
my_exec*      mysweep.inc*
borgc002:~/K_src>totalview ./my_exec
```



# Root Window

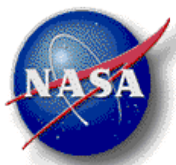


- State of all processes being debugged
- Process and Thread status
- Instant navigation access
- Sort and aggregate by status

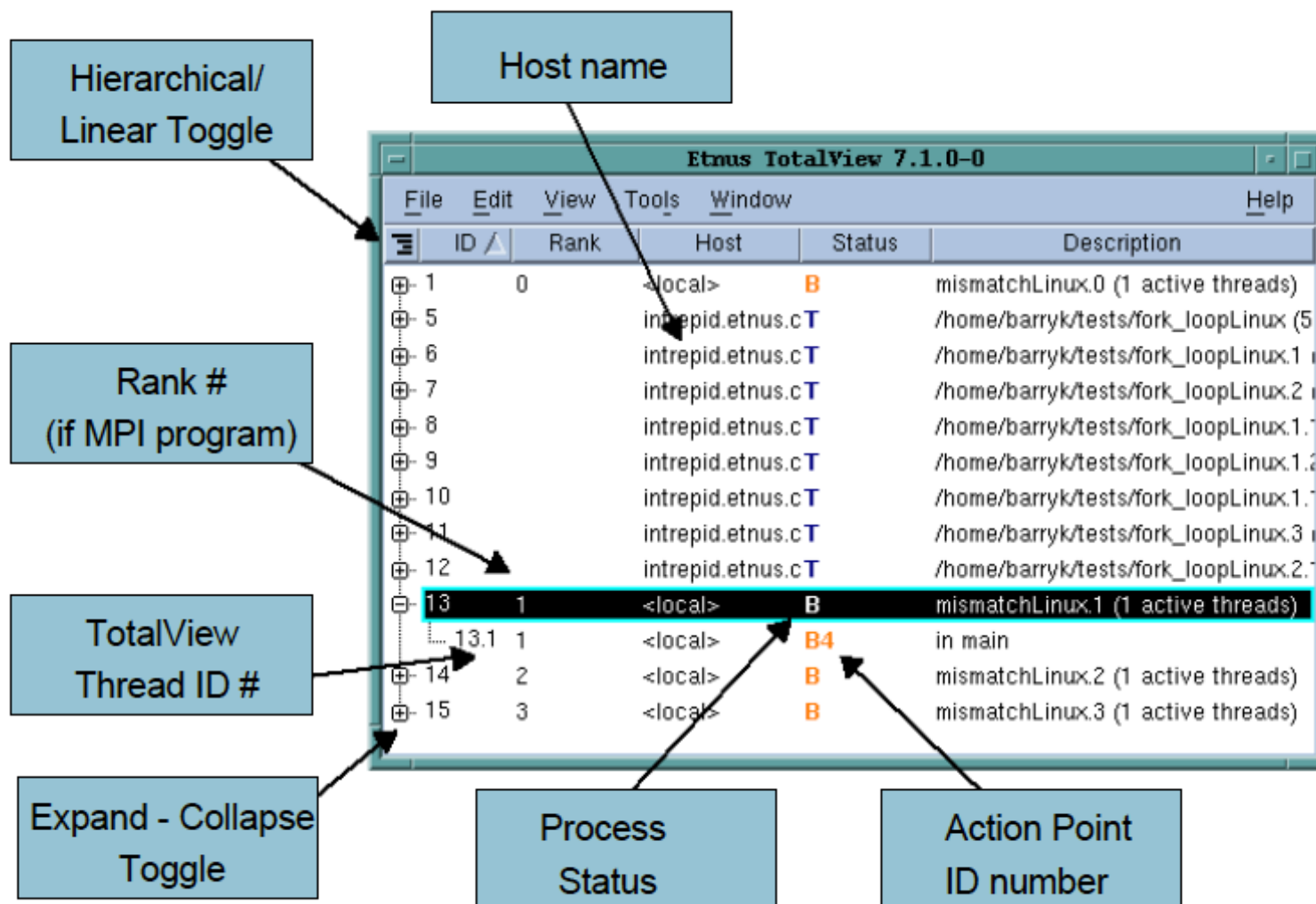
	ID	Rank	Host	Status
+	1	0	<local>	B
+	5		intrepid.etnus.c	T
+	6		intrepid.etnus.c	T
+	7		intrepid.etnus.c	T
+	8		intrepid.etnus.c	T
+	9		intrepid.etnus.c	T
+	10		intrepid.etnus.c	T
+	11		intrepid.etnus.c	T
+	12		intrepid.etnus.c	T
+	13	1	<local>	B
-	13.1	1	<local>	B4
+	14	2	<local>	B
+	15	3	<local>	B

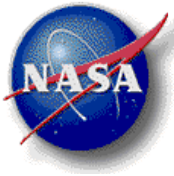
## TM Status Info

- T = stopped
- B = Breakpoint
- E = Error
- W = Watchpoint
- R = Running
- M = Mixed
- H = Held

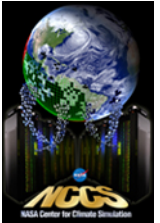


# Root Window





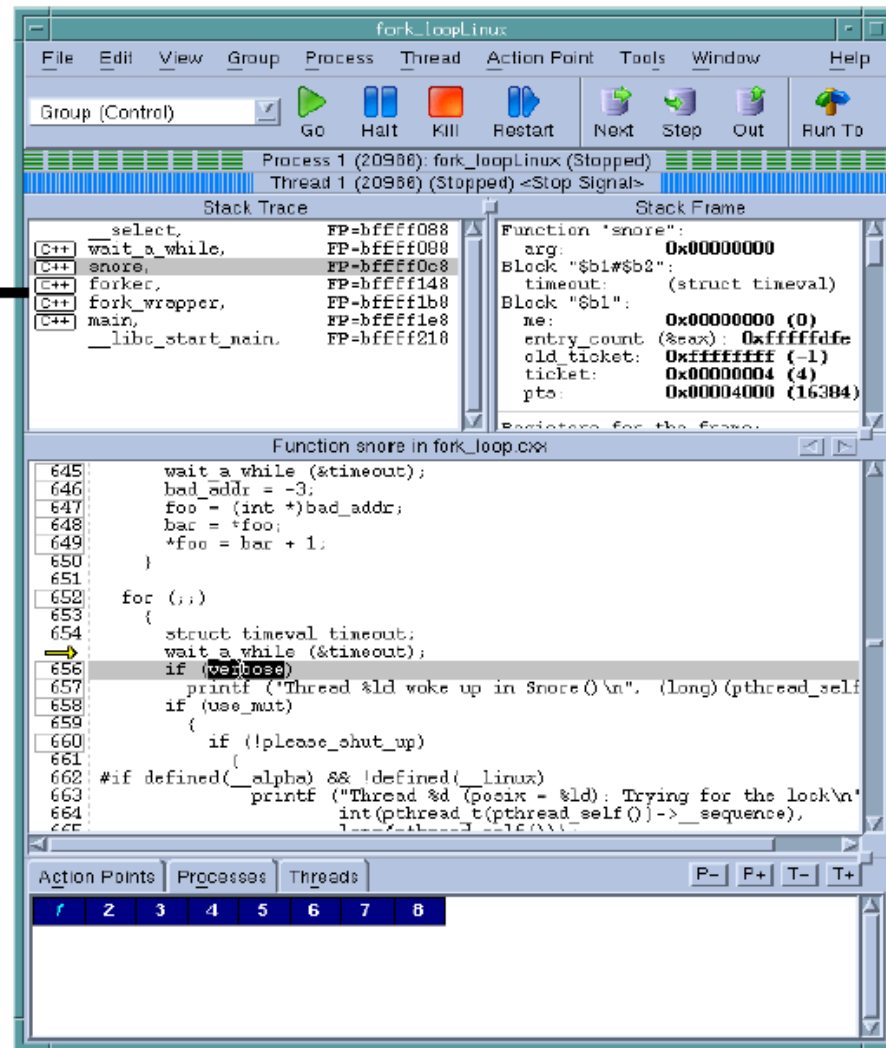
# Process Window



Stack Trace Pane

Provides detailed state of one process, or a single thread within a process

A single point of control for the process and other related processes

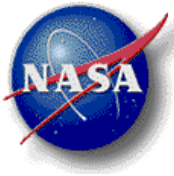


Toolbar

Stack Frame Pane

Source Pane

Tabbed Area



# Stack Trace and Stack Frame Panes



Process 1 (10893): fork\_loopLinux (At Breakpoint 1)  
Thread 1.1 (10893) (Stopped)

Stack Trace		Stack Frame
C++	wait_a_while,	FP=bffffeaa8
C++	snore,	FP=bffffeae8
C++	forker,	FP=bffffeb68
C++	fork_wrapper,	FP=bffffebd8
C++	main,	FP=bffffec08
C++	__libc_start_main,	FP=bffffec48

Function "wait\_a\_while":  
timeout: 0xbffffead0 -> (str  
Block "\$b2":  
result: 0x00000000 (0)

Registers for the frame:  
%eax: 0x00000000 (0)  
%ecx: 0x00000000 (0)  
%edx: 0x00000000 (0)  
%ebx: 0x401d7dd4 (1075674580)  
%esp: 0xbffffeaa0 (107274790)

Language

Name

Frame Pointer

Local Variables

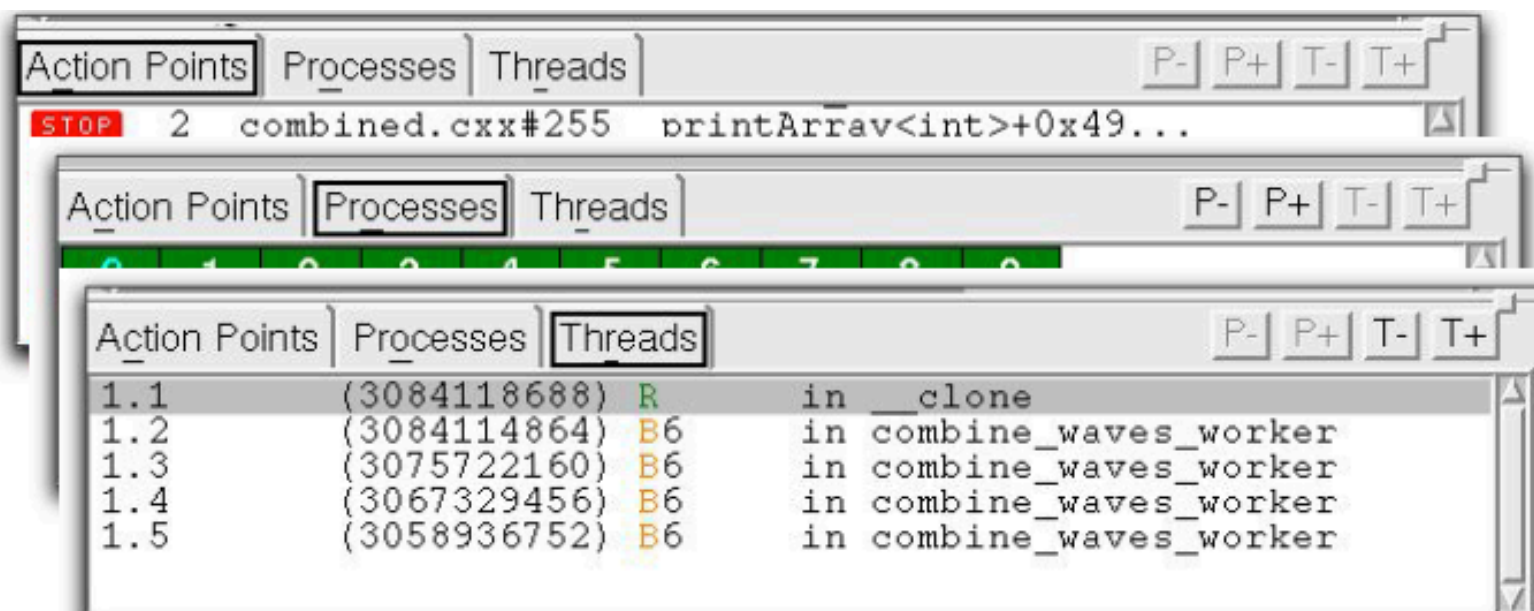
Register Values

- Click to refocus source pane

- Click to modify
- Dive for variable window



## Tabbed Area



### Action Points Tab

all currently defined  
action points

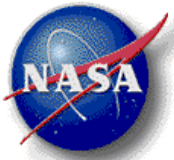
### Processes Tab

all current  
processes

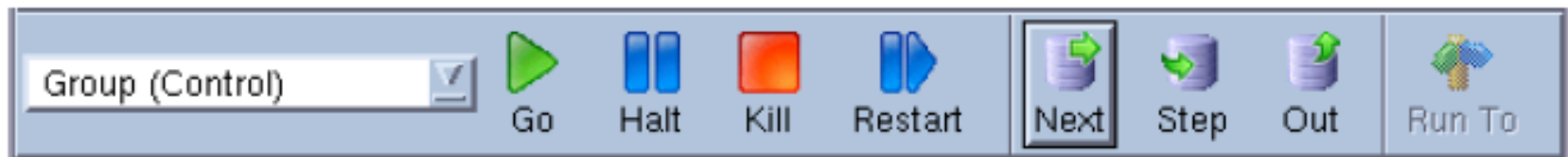
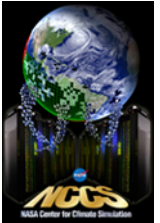
### Threads Tab:

all current threads,  
ID's, Status



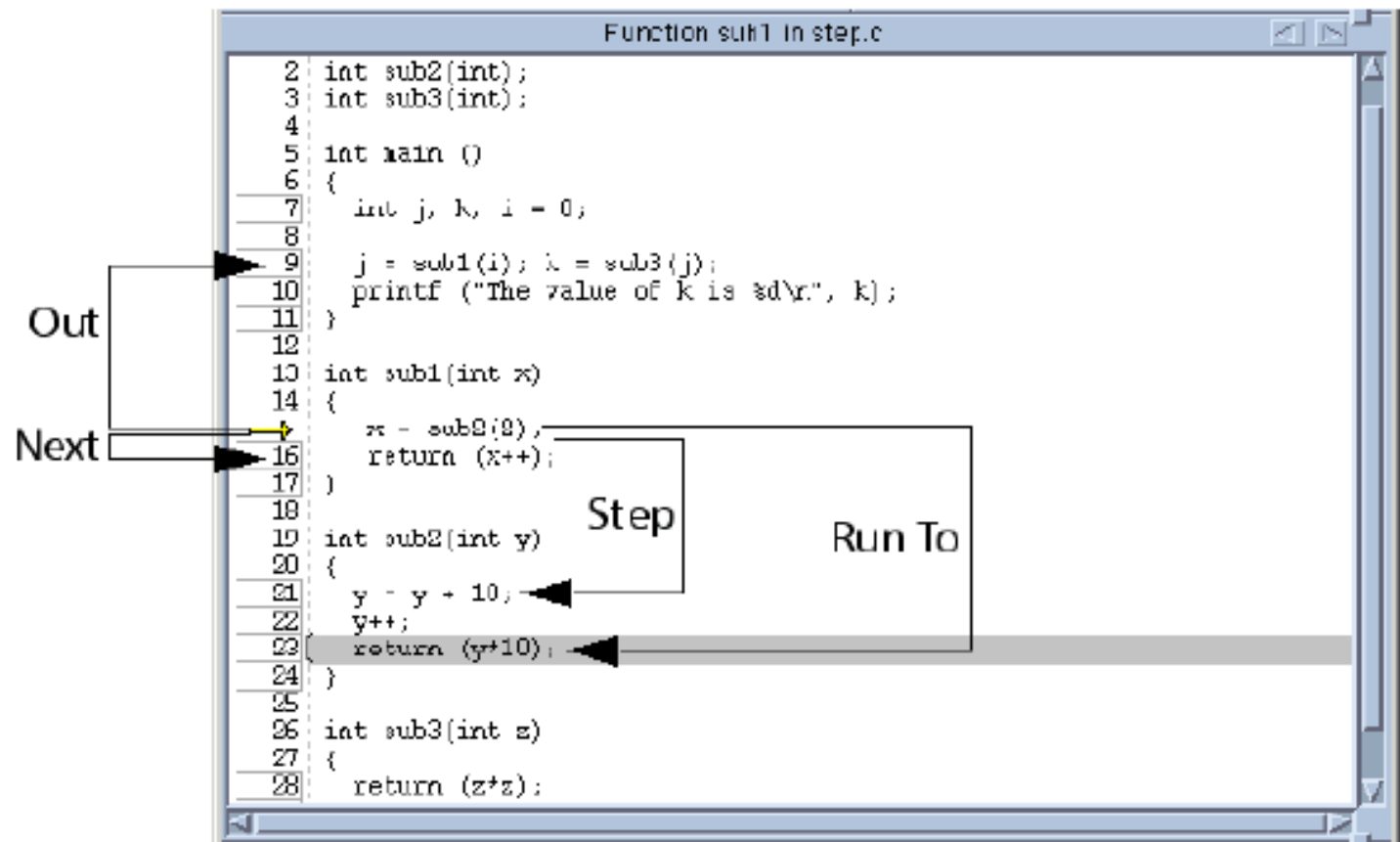


# Stepping Commands

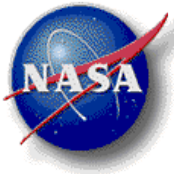


Based on  
PC location

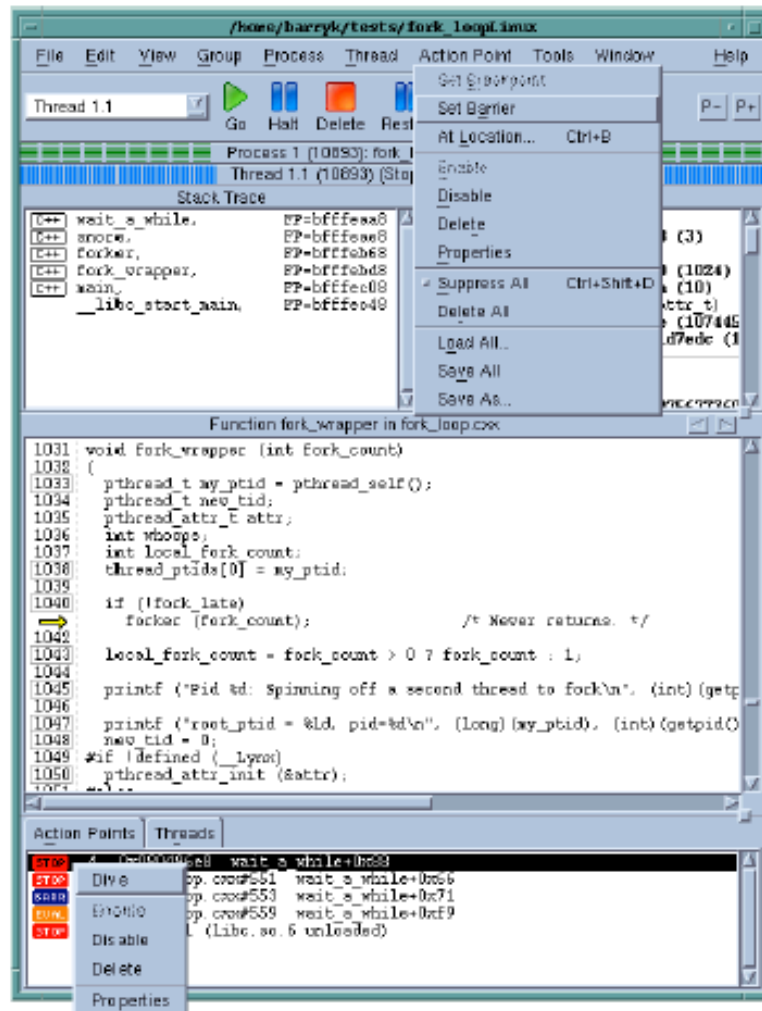
PC: Program  
Counter







# Action Points



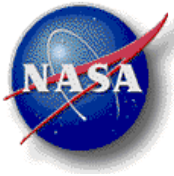
**Breakpoints**

**Barrier Points**

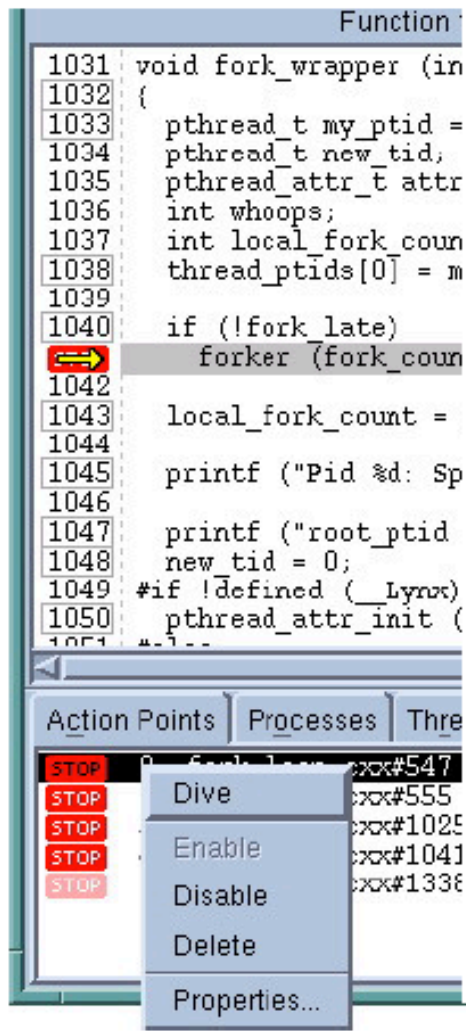
**Conditional Breakpoints**

**Evaluation Points**

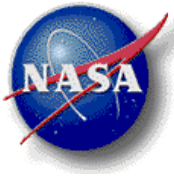
**Watchpoints**



# Setting Breakpoints



- Setting action points
  - Single-click line number
- Deleting action points
  - Single-click action point line
- Disabling action points
  - Single-click in Action Points Tab Pane
- Optional contextual menu access for all functions
- Action Points Tab
  - Lists all action points
  - Dive on an action point to focus it in source pane
- Action point properties
  - In Context menu
- Saving all action points
  - Action Point > Save All



# Conditional Breakpoint / Evalpoint

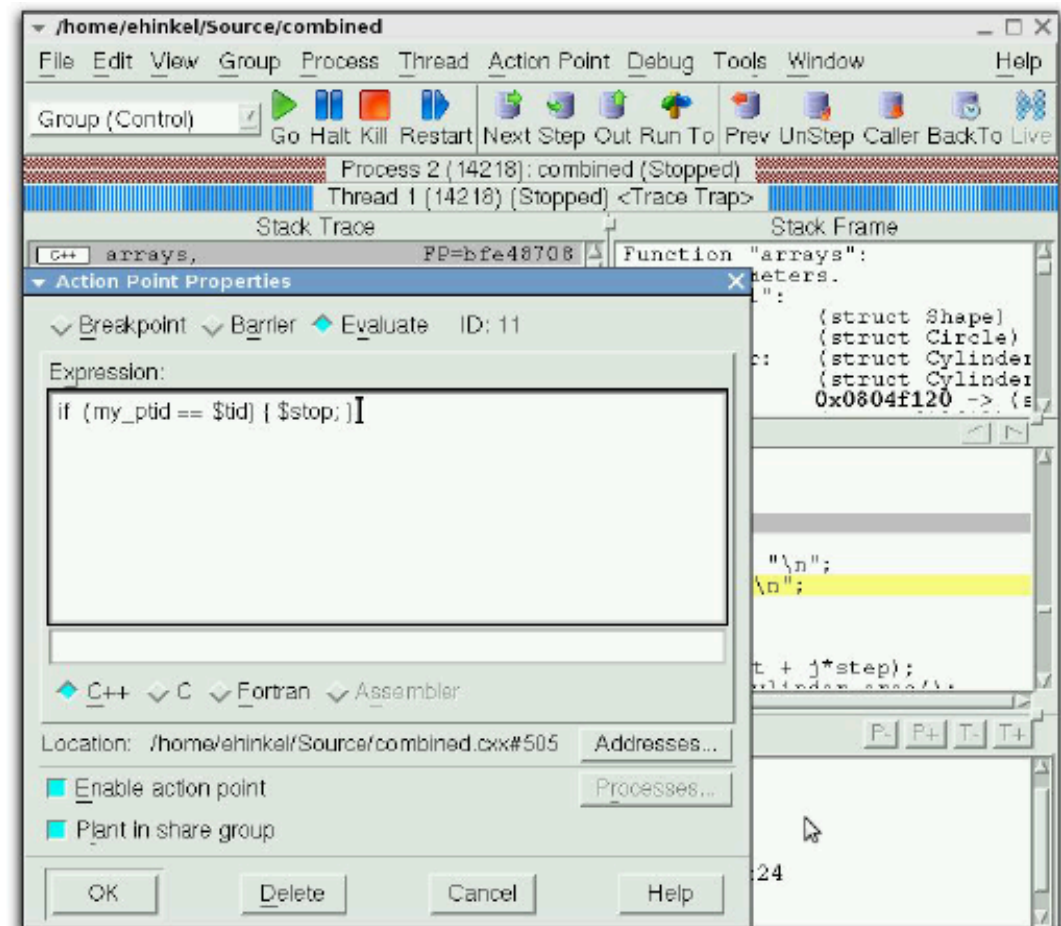


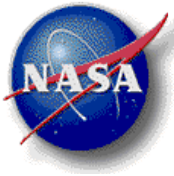
## Action Point → Properties

Evalpoints allow you to add a code fragment in Fortran or C/C++.

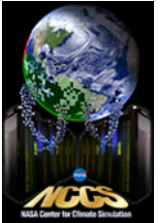
If the code fragment can make a decision whether to stop execution, it is called a conditional breakpoint.

Evalpoints can be used to test a fix, set values of variables, or visualize data automatically.



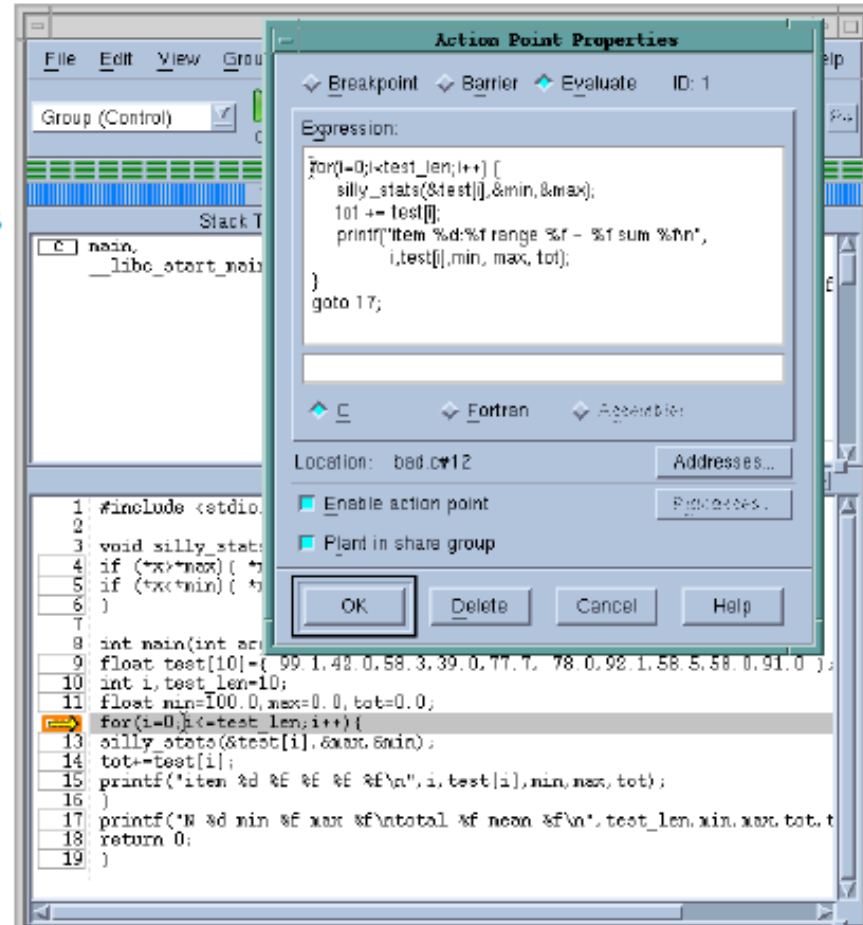


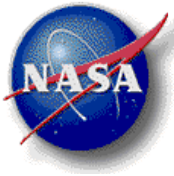
# Evalpoints Test Fixes on the Fly



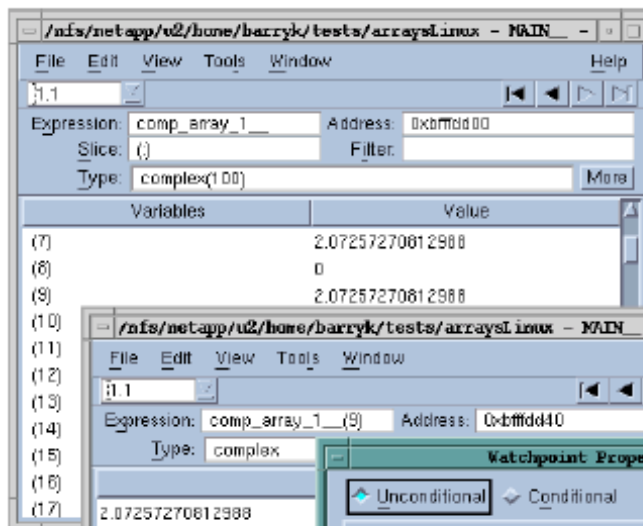
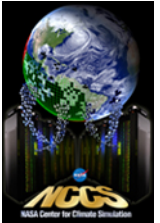
- Test small source code patches
- Call functions
- Set variables
- Test conditions
- C/C++ or Fortran
- Can't use C++ constructors
- Use program variables
- Can't modify variables or call functions with replay engine

```
Item 0:99.099998 range 99.099998 - 99.099998 sum 99.099998
Item 1:42.000000 range 42.000000 - 99.099998 sum 141.100006
Item 2:58.288998 range 42.000000 - 99.099998 sum 139.400008
Item 3:39.000000 range 39.000000 - 99.099998 sum 238.400008
Item 4:77.699997 range 39.000000 - 99.099998 sum 316.100006
Item 5:78.000000 range 39.000000 - 99.099998 sum 394.100006
Item 6:92.099998 range 39.000000 - 99.099998 sum 486.200012
Item 7:59.500000 range 39.000000 - 99.099998 sum 544.700012
Item 8:59.000000 range 39.000000 - 99.099998 sum 602.700012
Item 9:91.000000 range 39.000000 - 99.099998 sum 693.700012
N 10 min 39.000000 max 99.099998
total 693.700012 mean 69.370001
```





# Watchpoints

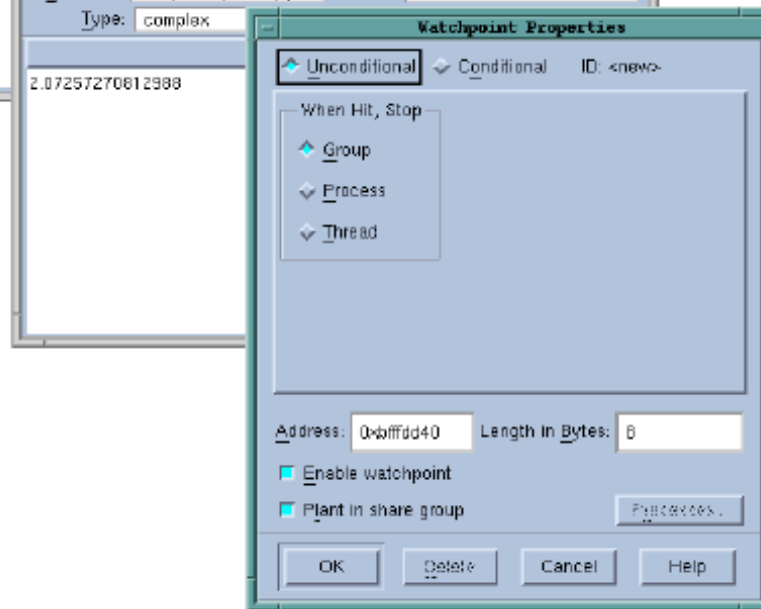


Watchpoints are set on a fixed memory region

Use *Tools > Watchpoint* from a Variable Window  
or

From source pane with contextual menu

When the contents of watched memory change, the watchpoint is triggered and TotalView stops the program.



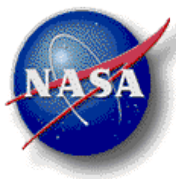
A Watchpoint tracks a memory location  
-- it does not trace a variable!

Watchpoints can be conditional or unconditional

TV variables *\$newval* and *\$oldval* can be used in the conditional expression

Uses Hardware Watchpoints with various limitations based on architecture





# Diving



## Items you dive on:

## Information Displayed:

Process or thread

When you dive on a thread in the Root Window, TotalView finds or opens a Process Window for that process. If it doesn't find a matching window, TotalView replaces the contents of an existing window and shows you the selected process.

Variable

The variable displays in a Variable Window.

Expression List Variable

Same as diving on a variable in the Source Pane: the variable displays in a Variable Window.

Routine in the Stack Trace Pane

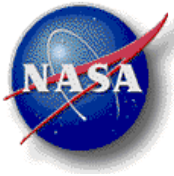
The stack frame and source code for the routine appear in a Process Window.

Array element, structure element, or referenced memory area

The contents of the element or memory area replace the contents that were in the Variable Window. This is known as a *nested* dive.

Pointer

TotalView dereferences the pointer and shows the result in a separate Variable Window. Given the nature of pointers, you may need to cast the result into the logical data type.



# Diving on Variables - Fortran Common Blocks



Diagram illustrating the process of diving into Fortran common blocks across four windows:

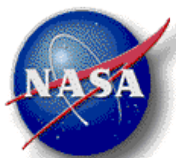
- Window 1: Stack Frame**
  - Variables: `INTEGER_ARRAY` (integer\*8(100)), `INTEGER_ARRAY_2` (integer\*2(100)), `MASTER_ARRAY` (integer(100)), `LOCAL_VAR` (0 (0x00000000)), `COMPLEX_ARRAY_2` (complex\*16(100)), `COMPLEX_ARRAY_1` (complex\*8(100)), `Common blocks` (Common).
- Window 2: iee - arraysAlpha - 1.1**
  - Expression: `iee` (Address: 0x1408214a0, Type: \$void).
  - Variables table:

Variables	Type	Value
<code>denoms</code>	integer(1000)	(integer(1000))
<code>iee_array</code>	integer(6)	(integer(6))
- Window 3: iee - iee - 1.1**
  - Expression: `denoms` (Address: 0x1408214b8, Type: integer(1000)).
  - Variables table:

Variables	Value
(1)	0 (0x00000000)
(2)	0 (0x00000000)
(3)	0 (0x00000000)
(4)	0 (0x00000000)
(5)	0 (0x00000000)
(6)	0 (0x00000000)
- Window 4: iee - iee - 1.1**
  - Expression: `iee_array` (Address: 0x1408214a0, Type: integer(6)).
  - Variables table:

Variables	Value
(1)	0 (0x00000000)
(2)	0 (0x00000000)
(3)	0 (0x00000000)
(4)	0 (0x00000000)
(5)	0 (0x00000000)
(6)	0 (0x00000000)

Arrows labeled "Dive" indicate the navigation path from the Stack Frame to the array window, then to the denoms window, and finally to the iee\_array window.



# Fortran 90 Modules



Tools → Fortran  
Modules

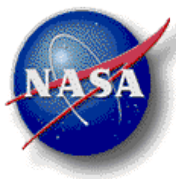
The screenshot displays the Fortran Modules tool interface. The main window, titled 'Fortran Modules', shows a list of modules from process '1' named 'modules'. The 'DATAMOD' module is selected. A sub-window, titled 'DATAMOD - modules - 1.1', provides details for this module. It shows the expression 'DATAMOD' and its address 'None'. Below this, a table lists the variables and their types and values.

Variables	Type	Value
a1	REAL*8(4)	(REAL*8(4))
v1	INTEGER*4	8 (0x00000008)
v2	INTEGER*4	0 (0x00000000)

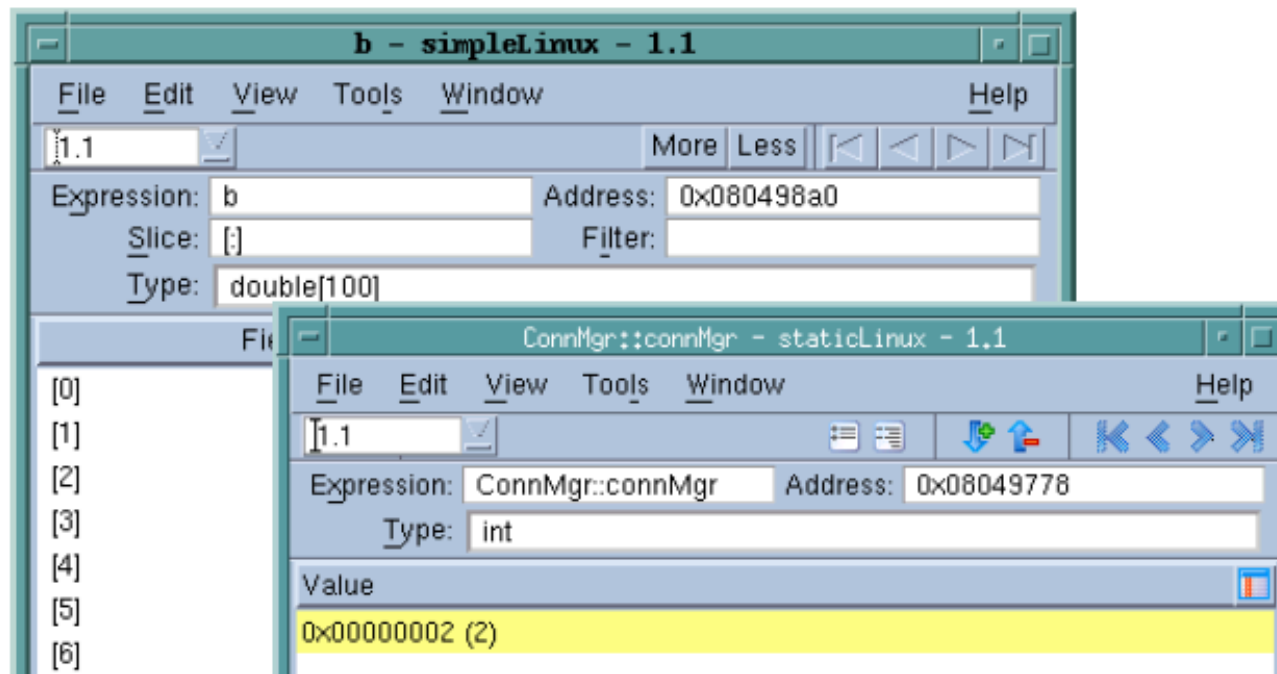
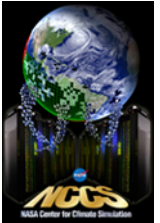
The background window shows the Fortran source code for the 'testmod' program, which uses the 'datamod' module. The code is as follows:

```
1 module datamod
2   integer v1
3   integer v2
4   real*8, dimension(4)
5 end module datamod
6
7 program testmod
8
9   use datamod
10
11   v1 = 8
12   v2 = 32
13
14   do i=1,4
15     a1(i) = 2*i
```





# The Variable Window



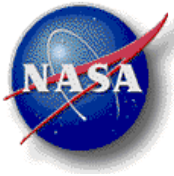
## Editing Variables

Window contents are updated  
automatically

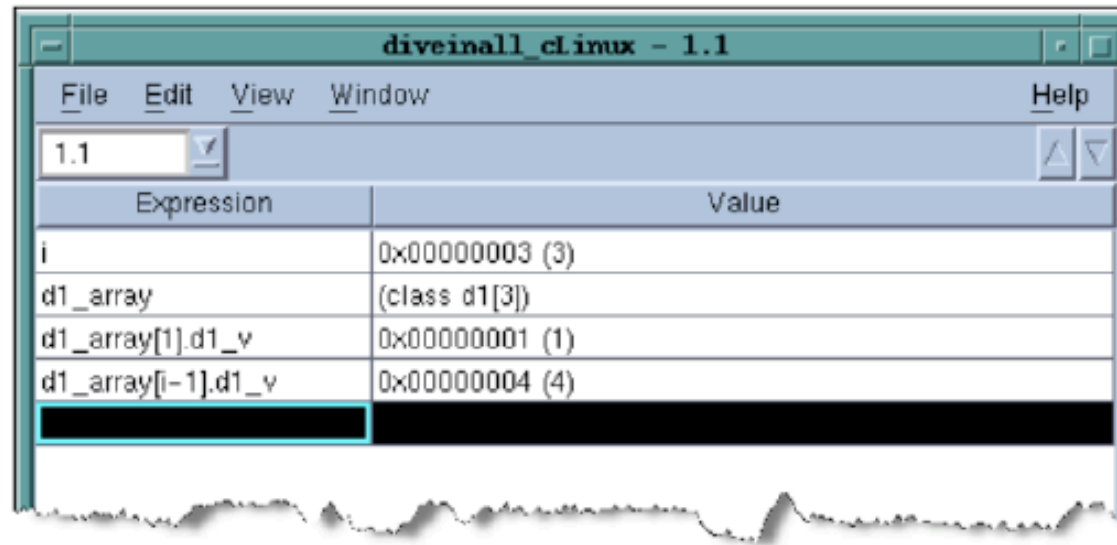
Changed values are highlighted

- Click once on the value
- Cursor switches into edit mode
- Esc key cancels editing
- Enter key commits a change
- Editing values changes the memory of the program

**NASA Center for Climate Simulation**



# Groups of Variables -- Expression List Window

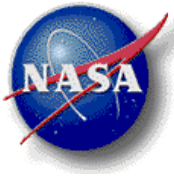


The screenshot shows a window titled "diveinall\_cLinux - 1.1" with a menu bar (File, Edit, View, Window, Help) and a version dropdown set to "1.1". Below is a table with two columns: "Expression" and "Value".

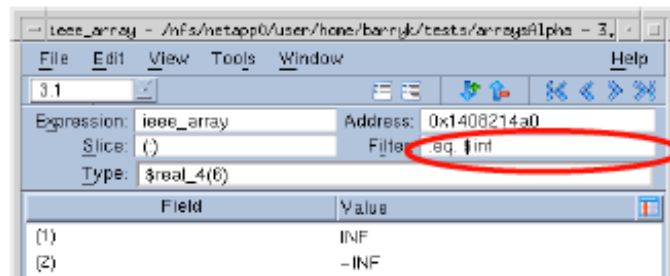
Expression	Value
i	0x00000003 (3)
d1_array	(class d1[3])
d1_array[1].d1_v	0x00000001 (1)
d1_array[i-1].d1_v	0x00000004 (4)

Add to the expression list using contextual menu with right-click on a variable, or by typing an expression directly in the window

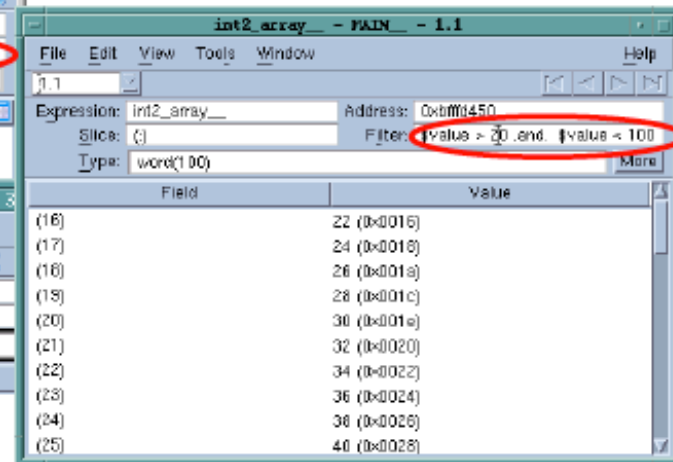
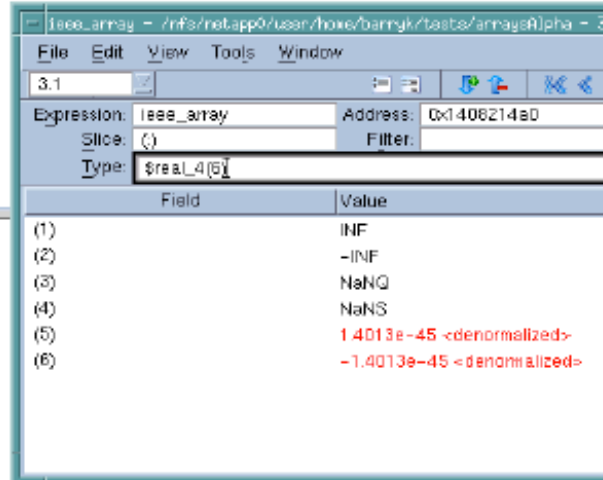
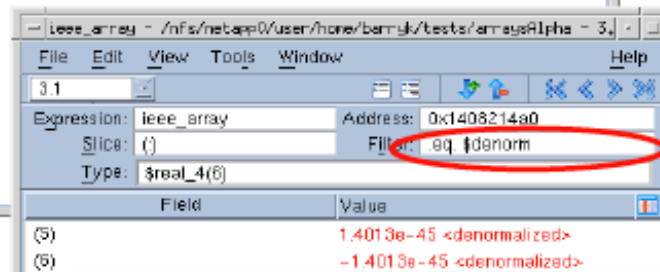
- Reorder, delete, add
- Sort the expressions
- Edit expressions in place
- Dive to get more info
- Updated automatically
- Expression-based
- Simple values/expressions
- View just the values you want to monitor

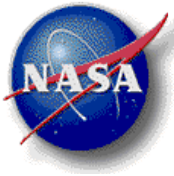


# Slicing and Filtering Arrays

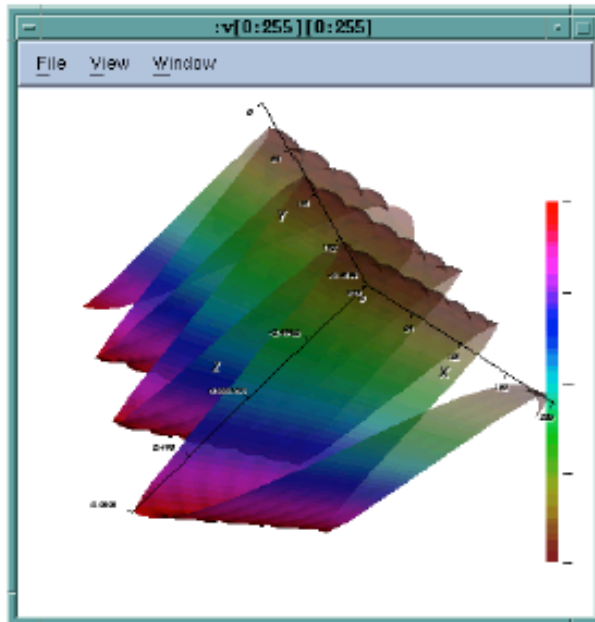


Slice notion is ([start:end:stride],  
[start:end:stride],...)

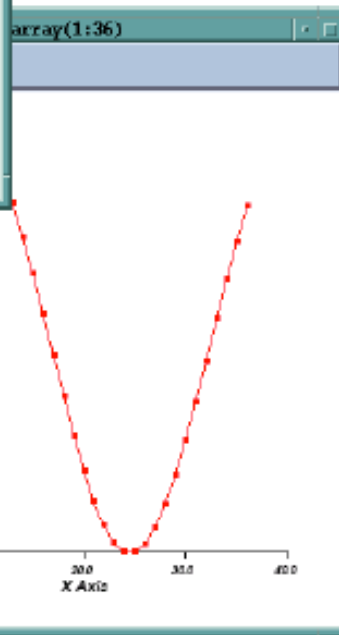




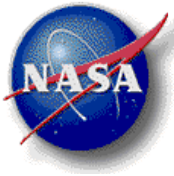
# Visualizing Arrays



- Visualize array data using Tools > Visualize from the Variable Window
- Large arrays can be sliced down to a reasonable size first
- Visualize is a standalone program
- Data can be piped out to other visualization tools



- Visualize allows to spin, zoom, etc.
- Data is not updated with Variable Window; You must revisualize
- `$visualize()` is a directive in the expression system, and can be used in evaluation point expressions.



# Variables Across Processes

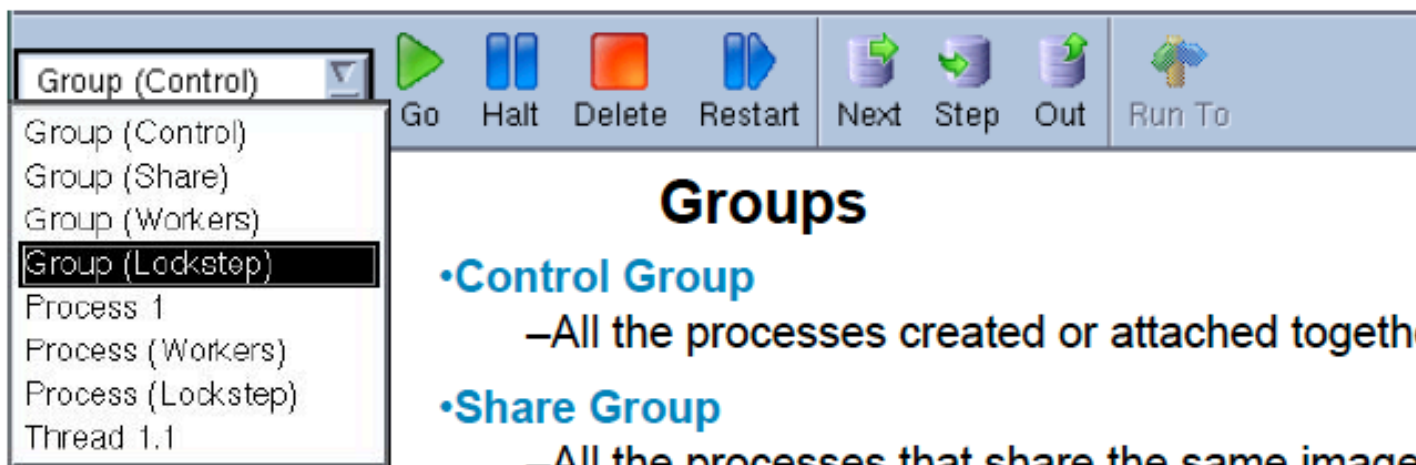


- **TotalView allows you to look at the value of a variable in all MPI processes**
  - Right Click on the variable
  - Select the View > View Across
- **TotalView creates an array indexed by process**
- **You can filter and visualize**
- **Use for viewing distributed arrays as well.**

Process	Value
mismatchAlpha.0	0x00000001 (1)
mismatchAlpha.1	0x00000000 (0)
mismatchAlpha.2	0x0000000c (12)
mismatchAlpha.3	0x0000000c (12)

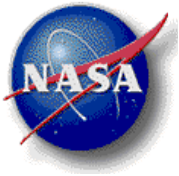


# Basic Process Control



## Groups

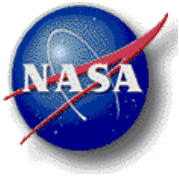
- **Control Group**
  - All the processes created or attached together
- **Share Group**
  - All the processes that share the same image
- **Workers Group**
  - All the worker threads within a control group. These threads can be in  $\geq 1$  share groups
- **Lockstep Group**
  - All threads at the same PC
- **Call Graph Group**
  - All processes going through the same node in the call graph
- **User Defined Group**
  - Process group defined in Custom Groups dialog



# Agenda



- Overview
- Basic Navigation and Control
- **Demo 1: Basic Navigation and Control**
  - ❖ Using an OpenMP Space weather application
- Debugging MPI Applications
- Demo 2: MPI Debugging:
  - ❖ Using GEOS5-AGCM

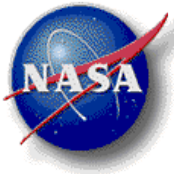


# Agenda



- Overview
- Basic Navigation and Control
- Demo 1: Basic Navigation and Control
  - ❖ Using an OpenMP Space weather application
- **Debugging MPI Applications**
- Demo 2: MPI Debugging:
  - ❖ Using GEOS5-AGCM

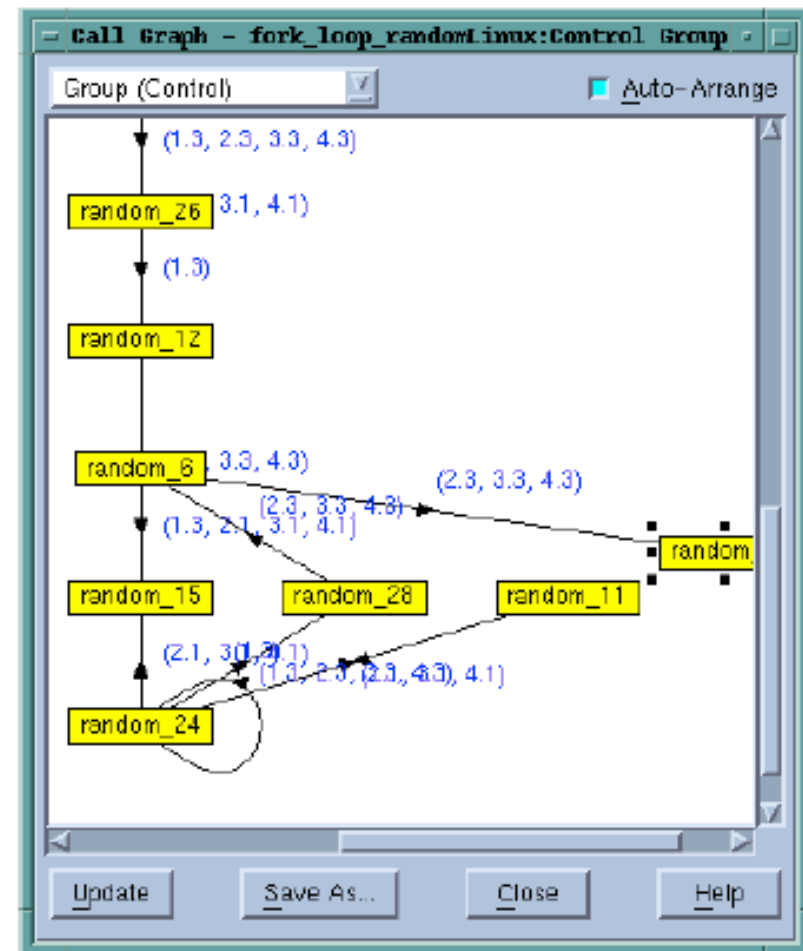




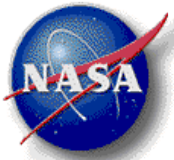
## Tools → Call Graphs



- **Quick view of program state**
  - Each call stack is a path
  - Functions are nodes
  - Numbers indicate which threads have a function on their call stack
- Construct process groups
- **Look for outliers**



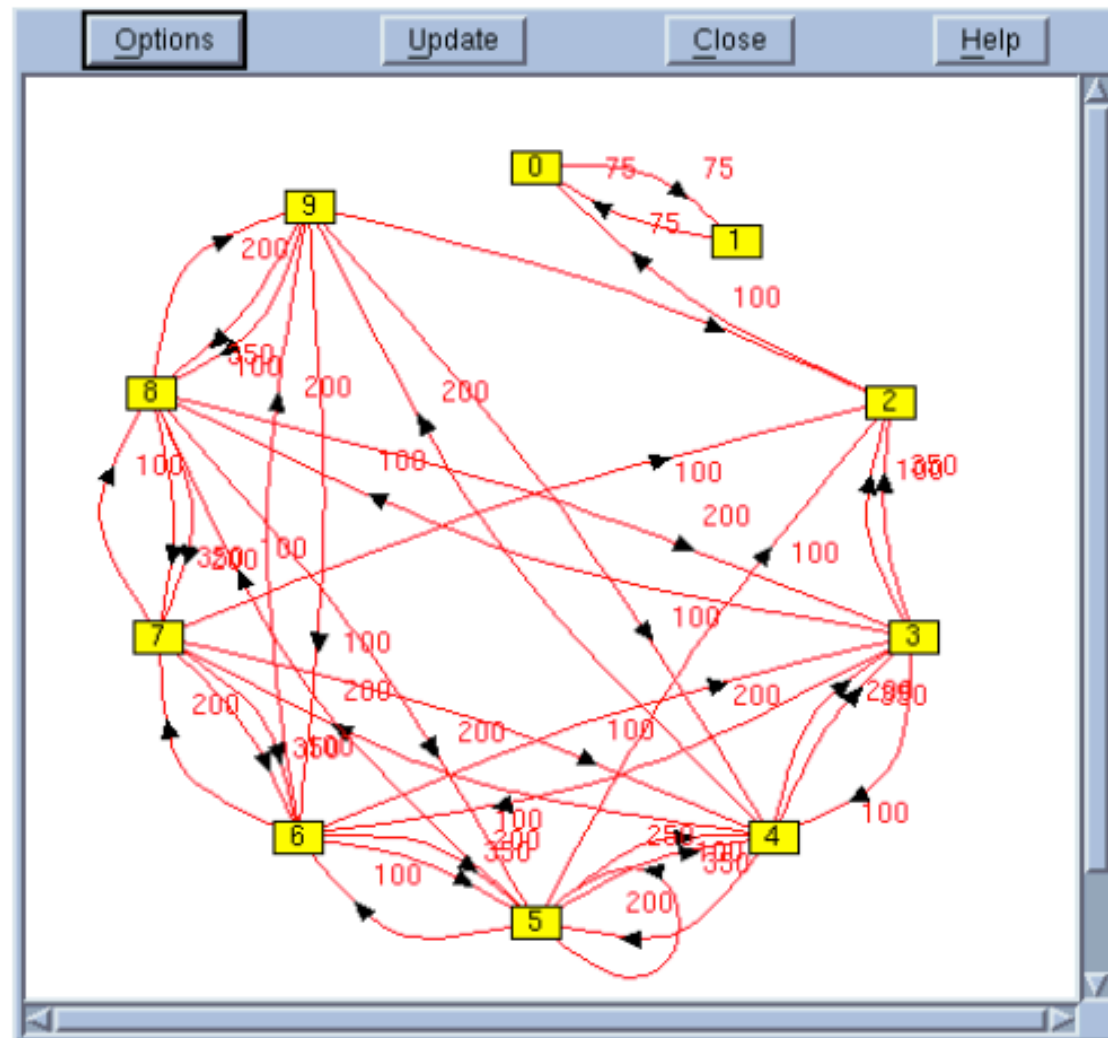
Dive on a node in the call graph to create a Call Graph group.



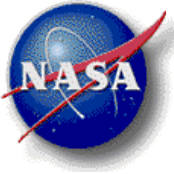
## Tools → Message Queue Graph



- **Hangs & Deadlocks**
- **Pending Messages**
  - Receives
  - Sends
  - Unexpected
- **Inspect**
  - Individual entries
- **Patterns**



NASA Center for Climate Simulation



# Tips for debugging MPI jobs

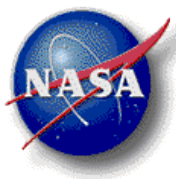


- **Reduce N**

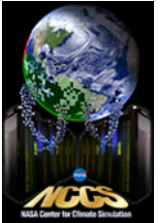
- **Problem:** Each process added requires overhead
- **Strategy:** Reduce the number of processes TotalView is attached to
  - Simply reducing N is best, however data or algorithm may require large N
- **Technique:** subset attach mechanism

- **Focus Effort**

- **Problem:** Some debugger operations are much more intensive than others, and when multiplied by N this could be significant
- **Strategy:** Reduce the interaction between the debugger and the processes
- **Technique:** Use TotalView's process control features to
  - Avoid single stepping - Usually causing Totalview hanging
  - Focus on one or a small set of processes

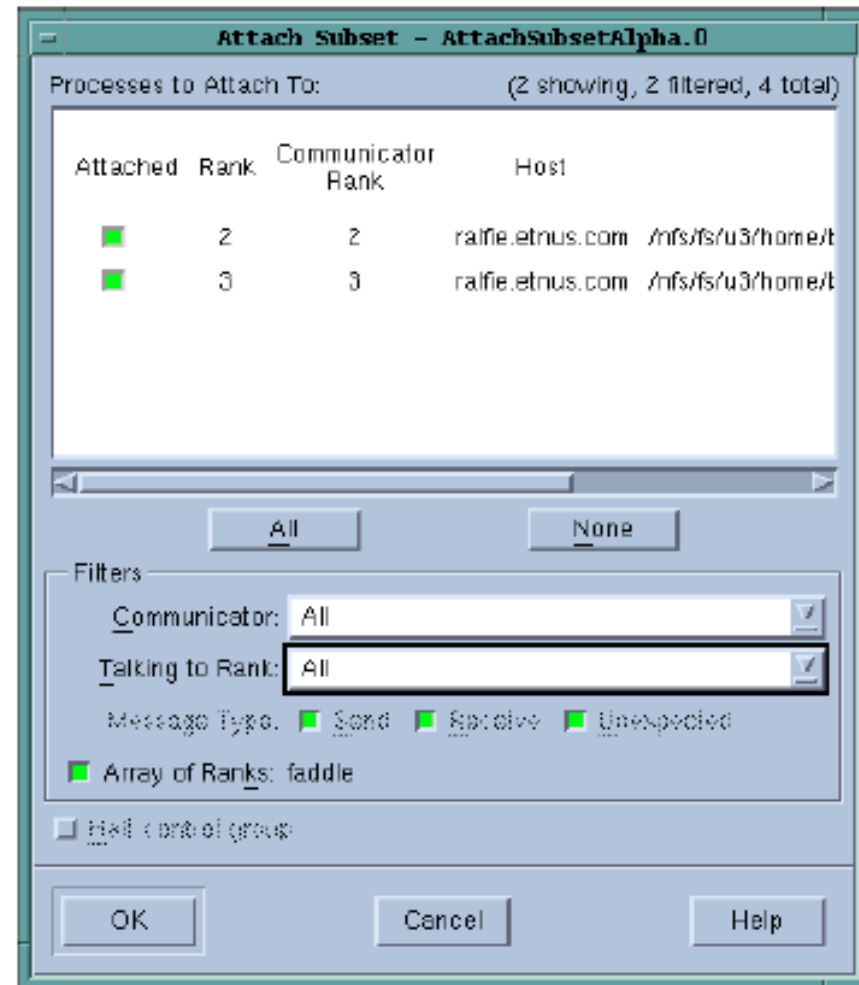


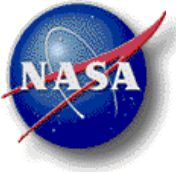
## Subset Attach



**TotalView does not need to be attached to the entire job**

- You can be attached to different subsets at different times through the run
- You can attach to a subset, run till you see trouble and then 'fan out' to look at more processes if necessary.
- This greatly reduces overhead
- It also requires a smaller license if you have a TotalView Team license.

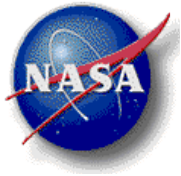




## Tips for debugging MPI jobs (Cont'd)



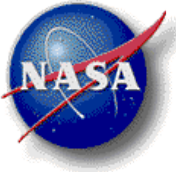
- Setting breakpoint behavior
- Synchronizing processes
  - ❖ Barrier and the process hold/release feature work together to control the executions
- Determining which processes/threads are executing
- Viewing variables across processes/threads
- Restarting from within Totalview



# Agenda



- Overview
- Basic Navigation and Control
- Demo 1: Basic Navigation and Control
  - ❖ Using an OpenMP Space weather application
- Debugging MPI Applications
- Demo 2: MPI Debugging:
  - ❖ Using GEOS5-AGCM



## Coming Up -- TotalView Part 2



- Replay Engine
  - ❖ Record and deterministic replay
  - ❖ Use breakpoints, watchpoints, and some conditional breakpoints when running in replay mode
- Memory Debugging
  - ❖ Why are memory bugs hard to detect?
  - ❖ How do your program's data reside in memory?
  - ❖ Heap graphic view
  - ❖ Leak and dangling pointer detection
  - ❖ Memory corruption report
  - ❖ Memory usage statistics